Theory of 2-3 Heaps

Tadao Takaoka

Department of Computer Science, University of Canterbury
Christchurch, New Zealand
tad@cosc.canterbury.ac.nz

Abstract. As an alternative to the Fibonacci heap, we design a new data structure called a 2-3 heap, which supports m decrease-key and insert operations, and n delete-min operations in $O(m + n \log n)$ time. The merit of the 2-3 heap is that it is conceptually simpler and easier to implement. The new data structure will have a wide application in graph algorithms.

1 Introduction

Since Fredman and Tarjan [6] published Fibonacci heaps in 1987, there has not been an alternative that can support n delete-min operations, and m decrease-key and insert operations in $O(m + n \log n)$ time. Logarithm here is with base 2, unless otherwise specified. Two representative application areas for these operations will be the single source shortest path problem and the minimum cost spanning tree problem. Direct use of these operations in Dijkstra's [5] and Prim's [7] algorithms with a Fibonacci heap will solve these two problems in $O(m + n \log n)$ time. A Fibonacci heap is a generalization of a binomial queue invented by Vuillemin [8]. When a key of a node v is decreased, the subtree rooted at v is removed and linked to another tree at the root level. If we perform this operation many times, the shape of a tree may become shallow, that is, the number of children from a node may become too many due to linkings without adjustment. If this happens at the root level, we will face a difficulty, when the node with the minimum is deleted and we need to find the next minimum. To prevent this situation, they allow loss of at most one child from any node. If one more loss is required, it will cause what is called cascading cut. This tolerance bound will prevent the degree of any node in the heap from getting more than $1.44 \log n$. The constant is the golden ratio derived from the Fibonacci sequence. Since this property will keep the degree in some bound, let us call this "horizontal balancing".

In the area of binary search trees, there are two well-known balanced tree schemes; the AVL tree [1] and the 2-3 tree [2]. When we insert or delete items into or from a binary search tree, we may lose the balance of the tree. To prevent this situation, we restore the balance by modifying the shape of the tree. As we control the path lengths, we can view this adjustment as vertical balancing. The AVL tree can maintain the tree height to be $1.44 \log n$ whereas the 2-3 tree will keep this to be $\log n$. As an alternative to the Fibonacci heap, we propose a

T. Asano et al. (Eds.): COCOON'99, LNCS 1627, pp. 41–50, 1999.

new data structure called a 2-3 heap, the idea of which is borrowed from the 2-3 tree and conceptually simpler than the Fibonacci heap. The degree of any node in the 2-3 heap is bounded by $\log n$, better than the Fibonacci heap by a constant factor. While the Fibonacci heap is based on binary linking, we base our 2-3 heap on ternary linking; we link three roots of three trees in increasing order according to the key values. We call this path of three nodes a trunk. We allow a trunk to shrink by one. If there is requirement of further shrink, we make adjustment by moving one or two subtrees from nearby positions. This adjustment may propagate, prompting the need for amortized analysis. The word "potential" used in [6] is misleading. It counts the number of nodes that lost one child. This number reflects some deficiency of the tree, not potential. Thus we use the word "deficit" for amortized analysis. We mark the trunk if it already lost one node and its corresponding tree. We define the deficit of the 2-3 heap to be the number of marked trunks. Amortized time for one decease-key or insert is shown to be $O(1)$, and that for delete-min to be $O(\log n)$.

The concept of r-ary linking is similar to the product of graphs. When we make the product of $G \times H$ of graphs G and H, we substitute H for every vertex in G and connect corresponding vertices of H if an edge exists in G. See Bondy and Murty [4], for example, for the definition. In the product of trees, only corresponding roots are connected. The 2-3 heap is constructed by ternary linking of trees repeatedly, that is, repeating the process of making the product of a linear tree and a tree of lower dimension. This general description of r-ary trees is given in Section 2. The precise definition of 2-3 heaps is given in Section 3. The description of operations on 2-3 heaps is given in Section 4. We also give amortized analysis of those operations. In Section 5, we consider several problems in implementation, and also some practical considerations for further speed up. Section 6 concludes this report. Note that our computational model is comparison-based. If we can use special properties of key values, there are efficient data structures, such as Radix-heaps [3].

2 Polynomial of trees

We define algebraic operations on trees. We deal with rooted trees in the following. A tree consists of nodes and branches, each branch connecting two nodes. The root of tree T is denoted by $root(T)$. A linear tree of size r is a liner list of r nodes such that its first element is regarded as the root and a branch exists from a node to the next. The linear tree of size r is expressed by bold face $\mathbf{r}$. Thus a single node is denoted by $\mathbf{1}$, which is an identity in our tree algebra. The emplty tree is nenoted by $\mathbf{0}$, which serves as the zero element. A product of two trees S and T, $P = ST$, is defined in such a way that every node of S is replaced by T and every branch in S connecting two nodes u and v now connects the roots of the trees substituted for u and v in S. Note that $\mathbf{2} * \mathbf{2} \neq \mathbf{4}$, for example, and also that $ST \neq TS$ in general. The symbol " $*$ " is used to avoid ambiguity.

The number of children of node v is called the degree of v and denoted by $deg(v)$. The degree of tree T, $deg(T)$, is defined by $deg(root(T))$. A sum of two